XAM Canonical Format

Finnbarr P. Murphy
(fpm@fpmurphy.com)

One of the key requirements for achieving long term data persistence is the ability to move data between archiving systems or, in the language of the SNIA `XAM (eXtensible Access Method)` specification, moving `XSets` between `XSystems`.

The `XAM` v1.0 specification supports this requirement by providing support for exporting and importing `Xsets`. It specifies the methods used to export an `XSet` from an `XSystem`, the resultant `XSet` canonical data interchange format (package) and the methods used to import an `Xset` into an `Xsystem`.

This post assumes that you are somewhat familiar with `XAM` and how to program to that specification using Java. It focuses on the format and content of the XSet canonical format package which consists of two main parts: an XML document which describes the policies, properties and streams of one or more XSets followed by the binary representation of the stream(s).

The package format conforms to the 2005 `W3C XML-binary Optimized Packaging (XOP)` recommendation. To quote from the recommendation:

> XOP define a general purpose serialization mechanism for the XML Infoset with binary content that is not only applicable to SOAP and MIME packaging, but to any XML Infoset and any packaging mechanism.

If you are unfamiliar with `XOP`, and most people are, an article by `Andrey Butov` in the December 2005 issue of `Doctor Dobb's Journal` contained a good introduction.

More than one `XSet` can be contained in a `package`. However the current `XAM SDK` reference implementation only supports one `XSet`. The XML document (AKA the *`XSet manifest`*) is a valid and well-formed `XML` document whose root element is *`xsets`*. It can be parsed and manipulated using `XSLT` and other `XML` tools. *`Annex B`* of the `XAM Architecture` document contains an `XML Schema Definition (XSD)` for the `XSet manifest`.

In order to study the package format in more detail, I wrote a small Java application called *`StoreHelloWorld`* which creates a new *`XSet`* containing two *`XStreams`*. The first *`Xstream`* contains the source code for the ubiquitous *`HelloWorld.java`* program. The second *`XStream`* contains the binary object *`HelloWorld.class`* encoded to `base64` and with a `MIME` type of *`application/base64`*. Normally you should not encode an `XStream` but displaying binary files in a blog is problematic and hence the workaround.

Here is the source code for *`StoreHelloWorld`*.

```
import java.io.BufferedOutputStream;
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
```



```
import java.io.InputStreamReader;
import java.io.InputStream;
import java.util.Calendar;

import org.snia.xam.XAMException;
import org.snia.xam.XAMLibrary;
import org.snia.xam.XSet;
import org.snia.xam.XStream;
import org.snia.xam.XSystem;
import org.snia.xam.XUID;
import org.snia.xam.util.XAMLibraryFactory;

public class StoreHelloWorld {
   private static XAMLibrary xamLib;
   private static XSystem xSystem;
   private static final int BUFFER_SIZE = 1024 * 128;

    private static XUID storeExportFile(String filename) throws XAMException {
        XUID xuid = null;

        System.out.println("Create XSet ....");
        XSet xSet = xSystem.createXSet(XSet.MODE_UNRESTRICTED);
        xSet.createProperty("app.filename", true, "HelloWorld.");

        try {
            System.out.println("Create XStream 1 ....");
           BufferedInputStream inputStream = new BufferedInputStream(new FileInputStream("
HelloWorld.java"));
            XStream xStream = xSet.createXStream("com.rhc.file.source", true, XAMLibrary.D
EFAULT_MIME_TYPE);

            byte[] buffer = new byte[BUFFER_SIZE];
            long bytesRead = 0;

            while((bytesRead = inputStream.read(buffer)) > 0) {
                long bytesPos = 0;
                while(bytesPos < bytesRead) {
long bytesWrite = xStream.write(buffer, bytesPos, bytesRead - bytesPos);
bytesPos += bytesWrite;
}
}
inputStream.close();
xStream.close();
} catch (FileNotFoundException fe) {
throw new IllegalArgumentException("Could not open HelloWorld.java for reading");
} catch (IOException ioe) {
System.err.println("Error reading from HelloWorld.java");
ioe.printStackTrace();
}

try {
System.out.println("Create XStream 2 ....");

InputStream inputStream = new Base64.InputStream(new FileInputStream("HelloWorld.class"),
Base64.ENCODE | Base64.DO_BREAK_LINES);
XStream xStream = xSet.createXStream("com.rhc.file.binary", true, "application/base64");

byte[] buffer = new byte[BUFFER_SIZE];
long bytesRead = 0;

while((bytesRead = inputStream.read(buffer)) > 0) {
                long bytesPos = 0;
                while(bytesPos < bytesRead) {
long bytesWrite = xStream.write(buffer, bytesPos, bytesRead - bytesPos);
bytesPos += bytesWrite;
}
}
inputStream.close();
```



```
xStream.close();
} catch (FileNotFoundException fe) {
throw new IllegalArgumentException("Could not open HelloWorld.class for reading");
} catch (IOException ioe) {
System.err.println("Error reading from HelloWorld.class");
ioe.printStackTrace();
}

System.out.println("Commit XSet .... ");
xuid = xSet.commit();
System.out.println("Assigned XUID: " + xuid.toString());

System.out.println("Close XSet .... ");
xSet.close();

File saveFile = new File(filename);
try {
System.out.println("\nOpen XSet .... ");
xSet = xSystem.openXSet(xuid, XSet.MODE_READ_ONLY);

BufferedOutputStream outputStream = new BufferedOutputStream(new
FileOutputStream(saveFile));

System.out.println("Open XStream for export ....");
XStream xStream = xSet.openExportXStream();

byte[] buffer = new byte[BUFFER_SIZE];
long bytesRead = 0;

System.out.println("Retrieve data .... ");
while((bytesRead = xStream.read(buffer)) > 0) {
               outputStream.write(buffer, 0, (int)bytesRead);
           }
           xStream.close();
           outputStream.close();

           System.out.println("Close XSet .... ");
           xSet.close();
       } catch (IOException ioe) {
            System.err.println("Error writing to " + saveFile);
            ioe.printStackTrace();
       }

       return (xuid);
  }

  public static void main(String[] args) {
      String xri = XRIstring.get();
      long exitCode = 0;

      InputStreamReader inputReader = new InputStreamReader(System.in);
      BufferedReader stdin = new BufferedReader(inputReader);

      try {
          xamLib = XAMLibraryFactory.newXAMLibrary();

          if (xri == null ) {
              System.out.print("Enter Xsystem address: ");
              String answer = stdin.readLine();
              if (!answer.equals(""))
                 xri = answer;
              else
                 System.exit(0);
          }

          System.out.println("Connecting to XSystem: " + xri );
          xSystem = xamLib.connect(xri);
```



```
            System.out.print("Name of export file: ");
            String filename = stdin.readLine();
            if (filename.equals(""))
                System.exit(0);

            XUID xuid = storeExportFile(filename);

            xSystem.close();
            System.out.println("\nClosed connection to XSystem");

            inputReader.close();
            stdin.close();

        } catch (XAMException xe) {
            exitCode = xe.getStatusCode();
            System.err.println(
                "XAM Error occured: " + xe.getMessage() + "(" + exitCode + ")");
        } catch (IOException e) {
            System.out.println("I/O Error occurred: " + e.getMessage());
            e.printStackTrace();
            exitCode = 1;
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
            exitCode = 1;
        }

        System.exit((int) exitCode);
    }
}
```

Rather than hard code an `XRI (XAM Resource Identifier)` in each `XAM` application, I use a separate Java class (*`XRIstring`*) to read in the `XRI` from the standard `XAM` properties file *`xam.properties`*.

I know that this is not kosher according to the current `XAM` specification but, in my humble opinion, this is the natural location for storing this string as most people will not be connecting simultaneously to multiple `XSystems`.

Here is the source for that Java class.

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.Properties;

public class XRIstring
{
   private static Properties props;
   private static String XRI = ".xam.config.xri";
   private static String xri = null;

   static {

       props = new Properties();
       try {
           props.load(new FileInputStream("xam.properties"));

           xri = props.getProperty(XRI);
           if (xri != null)
               System.out.println("XSystem Address: " + xri );

       } catch (FileNotFoundException e) {
          System.err.println("Could not find properties file. XSystem address must be manu
ally entered");
       } catch (IOException e) {
```



```
          System.err.println("Could not access properties file. XSystem address must be ma
nually entered");
       }
   }

   public static String get()
   {
        return(xri);
   }
}
```

This application also uses Robert Harder's popular *Base64* class which is available here. Assuming your CLASSPATH is set up correctly to find the XAM libraries, here is the output from *StoreHelloWorld* when you build and run it.

```
$ javac StoreHelloWorld.java XRIstring.java Base64.java

$ java StoreHelloWorld
Connecting to XSystem: snia-xam://centera_vim!XXX.XXX.XXX.XXX?/home/fpm/xam/xamconnect.pea
Name of export file: sample
Create XSet ....
Create XStream 1 ....
Create XStream 2 ....
Commit XSet ....
Assigned XUID: AAAEcwA9S8c0UTVHVDdRSDRSUU9VZUVWMkEyTTJGUUw2UUlHNDE0N1ZTMzFKTzFURkxWUjVKNkF
DVEo2Ng==
Close XSet ....

Open XSet for export ....
Retrieve data ....
Close XSet ....

Closed connection to XSystem
$
```

Here is the contents of the export file (*sample*) which was created.

```
Content-Type: Multipart/Related;boundary="___EMC_XOP_BOUNDARY_START___1245537694___EMC_XOP
_BOUNDARY_END___";
type="application/xop+xml";
start="";
start-info="text/xml"
Content-Description: EMC XSet Export Package

--___EMC_XOP_BOUNDARY_START___1245537694___EMC_XOP_BOUNDARY_END___
Content-Type: application/xop+xml; charset=UTF-8; type="text/xml"
Content-Transfer-Encoding: 8bit
Content-ID:


1.0.0



    default




      HelloWorld.


      2009-06-20T22:20:19.000Z
```



```
2009-06-20T22:20:21.000Z

2009-06-20T22:20:21.000Z

2009-06-20T22:20:21.000Z

AAAEcwA9S8c0UTVHVDdRSDRSUU9VZUVWMkEyTTJGUUw2UUlHNDE0N1ZTMzFKTzFURkxWUjVKNkFDVEo2Ng==

2009-06-20T22:20:21.000Z

base

true

0

2009-06-20T22:20:19.000Z

false

default

34372862-1dd2-11b2-aea0-f013836b5e75-10

true

4.0.672

34372862-1dd2-11b2-aea0-f013836b5e75-10

0

xam.binding

EMC Centera Binding XSet Clip

2009.06.20 22:20:19 GMT

2009.06.20 22:20:21 GMT

xam_challenge

xam_challenge
```



```
      2

      731

      6T90EGBR9LE5TFLVR5J6ACTJ66

      34372862-1dd2-11b2-aea0-f013836b5e75




      MD5

      3

      NULL

      4Q5GT7QH4RQOUeEV2A2M2FQL6QIG4147VS31JO1TFLVR5J6ACTJ66
```



```
--___EMC_XOP_BOUNDARY_START___1245537694___EMC_XOP_BOUNDARY_END___
Content-Type: text/text
Content-Transfer-Encoding: text
Content-ID:

Offset of AAAEcwA9S8c0UTVHVDdRSDRSUU9VZUVWMkEyTTJGUUw2UUlHNDE0N1ZTMzFKTzFURkxWUjVKNkFDVEo2
Ng==:com.rhc.file.source: 10955



Offset of AAAEcwA9S8c0UTVHVDdRSDRSUU9VZUVWMkEyTTJGUUw2UUlHNDE0N1ZTMzFKTzFURkxWUjVKNkFDVEo2
Ng==:com.rhc.file.binary: 11285



--___EMC_XOP_BOUNDARY_START___1245537694___EMC_XOP_BOUNDARY_END___
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: com.rhc.file.source

/*
 * HelloWorld.java
 *
```



```
 */

public class HelloWorld {
    public static void main(String[] args){
       System.out.println("Hello World!");
   }
}

--___EMC_XOP_BOUNDARY_START___1245537694___EMC_XOP_BOUNDARY_END___
Content-Type: application/base64
Content-Transfer-Encoding: binary
Content-ID: com.rhc.file.binary

yv66vgAAADIAHQoABgAPCQAQABEIABIKABMAFAcAFQcAFgEABjxpbml0PgEAAygpVgEABENvZGUB
AA9MaW5lTnVtYmVyVGFibGUBAARtYWluAQAWKFtMamF2YS9sYW5nL1N0cmluZzspVgEAClNvdXJj
ZUZpbGUBAA9IZWxsb1dvcmxkLmphdmEMAAcACAcAFwwAGAAZAQAMSGVsbG8gV29ybGQhBwAaDAAb
ABwBAApIZWxsb1dvcmxkAQAQamF2YS9sYW5nL09iamVjdAEAEGphdmEvbGFuZy9TeXN0ZW0BAANv
dXQBABVMamF2YS9pby9QcmludFN0cmVhbTsBABNqYXZhL2lvL1ByaW50U3RyZWFtAQAHcHJpbnRs
bgEAFShMamF2YS9sYW5nL1N0cmluZzspVgAhAAUABgAAAAAAAgABAAcACAABAAkAAAAdAAEAAQAA
AAUqtwABsQAAAAEACgAAAAYAAQAAAAYACQALAAwAAQAJAAAAJQACAAEAAAAJsgACEgO2AASxAAAA
AQAKAAAACgACAAAACAAIAAkAAQANAAAAAgAO
--___EMC_XOP_BOUNDARY_START___1245537694___EMC_XOP_BOUNDARY_END___--
```

As you can see this export package is fairly large even though I am only storing two small `XStreams` in the `XSet` which I created.

The `XSet manifest` is everything between the and tags. It appears to be a valid and well formed XML document although I have not formally checked it.

The first section, i.e. the section, lists the `XSystem` policies that are in effect for the included `XSet(s)` . Here the policy is set to the default policy for the `Xsystem` from which I exported the `Xset`.

It is followed by a section whose top element is which details the properties and `XStream(s)` for a single `XSet`. Each `XSet` property element contains the following attributes: *`type`*, *`readonly`*, *`binding`* and *`length`*. Each `XStream` is mapped to an element which contains a child element which contains the *`content-id`* to the corresponding `MIME` part containing the actual contents of the `XStream`.

As you can see the actual contents of an `XStream` is included as a `MIME` attachment after the `XSet manifest`. As an aside, note that there is no requirement for an `XStream` to verify that its contents actually match the declared `MIME` type.

The first `MIME` attachment shown above (AKA the *`root`* `MIME`) lists the offsets of each of the two `XStreams`. If an `XSet` does not contain an `XStream`, then there are no `MIME` attachments. Other than the root `MIME` attachment the order of the `MIME` attachments is not significant. The `MIME` attachments conform to the requirements of the `MIME` Multipart/Related Content-type specification (`RFC 2387`).

Here is the export package generated when *`StoreHelloWolld`* was targeted at the `SNIA XAM` reference `VIM (Vendor Interface Module)`.

```
Content-Type: Multipart/Related;boundary="--SNIA_REFERENCE_VIM_MIME_BOUNDARY_2009-06-22T21:
13:44.330-05:00.XZY_+";type="application/xop+xml";start="";start-info="text/xml"
Content-Description:Export of XSet: AAA6AwAePKwxMjQ1NzIzMjI0Mjc4ATTLJDRoe2iJ

----SNIA_REFERENCE_VIM_MIME_BOUNDARY_2009-06-22T21:13:44.330-05:00.XZY_+

Content-Type:application/xop+xml;    charset="UTF-8";    type="text/xml"
Content-Transfer-Encoding:8bit
Content-ID:
```



```
1.0.0


        false

        .org.snia.refvim.default.mgmt.policy

        true

        2009-06-22T21:13:44.276-05:00

        base

        event

        2009-06-22T21:13:44.276-05:00

        2009-06-22T21:13:44.276-05:00

        2009-06-22T21:13:44.225-05:00

        2009-06-22T21:13:44.276-05:00

        AAA6AwAePKwxMjQ1NzIzMjI0Mjc4ATTLJDRoe2iJ

        HelloWorld.



        XStream_8508062132085140723.data


        XStream_3869980320333904649.data




----SNIA_REFERENCE_VIM_MIME_BOUNDARY_2009-06-22T21:13:44.330-05:00.XZY_+

Content-Type:text/text
Content-Transfer-Encoding:text
Content-ID:
Offset of AAA6AwAePKwxMjQ1NzIzMjI0Mjc4ATTLJDRoe2iJ:com.rhc.file.binary@snia.org:3948
Offset of AAA6AwAePKwxMjQ1NzIzMjI0Mjc4ATTLJDRoe2iJ:com.rhc.file.source@snia.org:4706

----SNIA_REFERENCE_VIM_MIME_BOUNDARY_2009-06-22T21:13:44.330-05:00.XZY_+

Content-Type:application/base64
Content-Transfer-Encoding:binary
Content-ID:

yv66vgAAADIAHQoABgAPCQAQABEIABIKABMAFAcAFQcAFgEABjxpbml0PgEAAygpVgEABENvZGUB
AA9MaW5lTnVtYmVyVGFibGUBAARtYWluAQAWKFtMamF2YS9sYW5nL1N0cmluZzspVgEAClNvdXJj
ZUZpbGUBAA9IZWxsb1dvcmxkLmphdmEMAAcACAcAFwwAGAAZAQAMSGVsbG8gV29ybGQhBwAaDAAb
ABwBAApIZWxsb1dvcmxkAQAQamF2YS9sYW5nL09iamVjdAEAEGphdmEvbGFuZy9TeXN0ZW0BAANv
dXQBABVMamF2YS9pby9QcmludFN0cmVhbTsBABNqYXZhL2lvL1ByaW50U3RyZWFtAQAHcHJpbnRs
bgEAFShMamF2YS9sYW5nL1N0cmluZzspVgAhAAUABgAAAAAAAgABAAcACAABAAkAAAAdAAEAAQAA
AAUqtwABsQAAAAEACgAAAAYAAQAAAAYACQALAAwAAQAJAAAAJQACAAEAAAAJsgACEgO2AASxAAAA
AQAKAAAACgACAAAACAAIAAkAAQANAAAAAgAO

----SNIA_REFERENCE_VIM_MIME_BOUNDARY_2009-06-22T21:13:44.330-05:00.XZY_+

Content-Type:application/octet-stream
Content-Transfer-Encoding:binary
Content-ID:

/*
 * HelloWorld.java
```



```
 *
 */

public class HelloWorld {
    public static void main(String[] args){
       System.out.println("Hello World!");
   }
}

----SNIA_REFERENCE_VIM_MIME_BOUNDARY_2009-06-22T21:13:44.330-05:00.XZY_+--
```

As you can see this export package is considerably more compact then the previous export package generated when I used the `EMC Centera XAM VIM`.

Well, that is about all for now. Hopefully this post was useful to you if you are just coming up to speed on `XAM` and wondering how to import and export `XSets`.

BTW, Happy Father's Day to all dads who may be reading this post.

